Invited Paper

Progressive Representation, Transmission, and Visualization of 3D Objects

Masahiro Okuda and Tsuhan Chen

Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Tel: +1 (412) 268-7536 Email: tsuhan@cmu.edu

Keywords: 3D data, 3D models, 3D meshes, geometry coding, texture coding, progressive coding, progressive transmission, streaming perceptual quality, visualization

ABSTRACT

Files containing 3D objects, typically represented as 3D meshes with certain geometry and texture information, are very large. Therefore, not only do 3D objects take a lot of storage space, it is also extremely time-consuming to transmit them over the network for visualization. In addition, most 3D visualization applications need the entire 3D data file to render the 3D object even though the user may be interested in only a small part or a low-resolution version of the object. Progressive coding of 3D objects can resolve these problems. In this paper, we report our recent progress in progressive representation, transmission, and visualization of 3D objects. In our scheme, both geometry and the texture of the 3D object are progressively coded and transmitted. More perceptually important information is transmitted before the less important information, which allows the user to stop the transmission at any time and yet retain the best available perceptual quality of the object at that time. Furthermore, the visible portion of the object is transmitted first and the non-visible portion is transmitted later, or not transmitted at all, in order to save the overall bandwidth.

1. INTRODUCTION

Computer graphics using 3D objects are becoming more and more popular in many applications including movie productions, TV commercials, and video games. However, files containing 3D objects still remain to be very large, so it is time-consuming to retrieve 3D objects from the storage device or to download them from the network. Moreover, most 3D visualization applications have to obtain the entire file of a 3D model in order to display the model, even when the user is interested only in a small part, or a low-resolution version, of the model. This makes it very ineffective when 3D models need to be shared or transmitted over the network. Therefore, progressive representation of 3D objects is desired to solve these problems to meet various needs [1],[2],[3].

One existing scheme for 3D object representation is the compressed binary format specified by the Web3D Consortium, a group that standardizes formats of files containing 3D models [4]. Another example is the work recently done by the Synthetic/Natural Hybrid Coding (SNHC) subgroup with MPEG-4 [5],[6]. A 3D model generally contains geometry, texture, and other attribute data like normals and colors. Most of existing progressive coding algorithms focus only on the geometry. These algorithms are progressive either in terms of *resolution*, i.e., the number of vertices [7],[8], or in terms of the signal-to-noise ratio (SNR), i.e., the accuracy of vertex coordinates [9],[10],[11]. However, most existing schemes provide only a limited number of levels of detail (LOD). Furthermore, they consider only coding of the geometry information. Once some vertices and triangles are decimated during the simplification process, the corresponding attribute data are also discarded. To make the simplified model look realistic and similar to the original model, simplification of the attribute data including normals, colors and textures, should be taken into account together with the geometry.

In this paper, we propose a joint geometry/texture coding scheme for arbitrary manifold 3D models resulting in progressive bitstreams. The proposed algorithm is based on a vertex decimation approach. The 3D model is transmitted vertex-by-vertex,

In *Input/Output and Imaging Technologies II*, Yung-Sheng Liu, Thomas S. Huang, Editors, Proceedings of SPIE Vol. 4080 (2000) • 0277-786X/00/$15.00

providing "granular" progressive transmission. The textures are also coded progressively, and texture bits and geometry bits are combined into one bitstream. The proposed method allows the user to stop the transmission at any time and yet obtain the best available quality in terms of both geometry and texture. In addition to providing joint geometry/texture progressive coding, the proposed scheme is also comparable to or better than other existing schemes in terms of coding efficiency.

2. THE PROPOSED SCHEME

There are two ways to correspond attribute data with the 3D model. One is to associate attribute data to each vertex. In this paper, this type of data is called "vertex attributes" (vertex attributed texture coordinates, vertex attributed colors, and so on). The other to associate attribute data to each *corner*, i.e., each vertex in each triangle. We call this type of data "corner attributes" (corner attributed texture coordinates, corner attributed colors, and so on).

In the proposed scheme, the encoder removes vertices until we are left with a base mesh that has only a small fraction of the vertices and triangles in the original model. To send a 3D model progressively, we start by sending the vertex positions and vertex indices of the base mesh. Then, enhancements to this base mesh are sent, vertex-by-vertex, until all the vertices are transmitted. In the mean time, texture information is transmitted progressively, as detailed in the next section.

2.1. Vertex Decimation

The process of vertex decimation is as follows. In a triangular mesh, there is a ring of triangles that surround every vertex, as illustrated in Figure 2. To ensure that the most perceptually important vertices are sent before the less important vertices, we need a method for measuring the perceptual importance of vertices. We introduce two measures. The measure $v(i)$ is defined as the difference in the volume caused by the decimation, by forming tetrahedrons with the removed vertex as the apex and the new triangles as the base, and adding up the volumes of these tetrahedrons. We can see that the measure $v(i)$ is similar to the "curvature" of the mesh at the vertex.

The other measure $c(i)$ is to quantify texture similarity in the surrounding triangles. During the simplification procedure, some triangles are decimated. In case that each triangle has a corresponding color or small texture, some colored triangles or textures are lost by the simplification. Thus, in order to preserve the appearance of the models, we need to carefully choose the vertices being decimated. In this paper, we adopt the color difference as the criterion. The idea of this second measure is as follows. Consider two vertices on 3D surfaces, $\mathbf{v}_1$ and $\mathbf{v}_2$ in Figure 1. While the triangles connected to $\mathbf{v}_1$ have different color information, those of $\mathbf{v}_2$ have similar colors. In order to preserve the appearance of the models, $\mathbf{v}_2$ should be decimated prior to $\mathbf{v}_1$ even when the measure $v(i)$ may be the same for these two cases. The other measure $c(i)$ therefore seeks to find the importance of vertices in terms of the color difference. In our current implementation, we first transform RGB values to the LUV color space. Next we find the average of each component, and use the sum of the absolute differences between the averages as the measure.

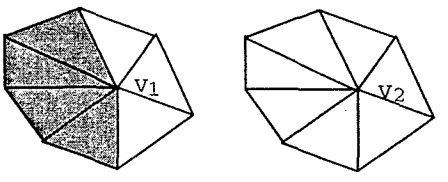


Figure 1: Left: triangles with different colors. Right: triangles with similar colors

To evaluate the overall perceptual importance of each vertex, we calculate the weighted sum of the two measures as follows.

$$m(i) = \alpha v(i) + (1-\alpha)c(i) \qquad (1)$$

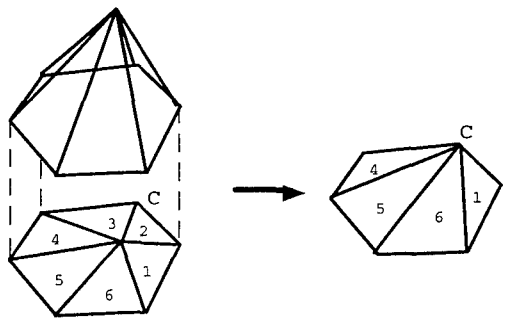
Vertices with large $m(i)$ are considered more perceptually important. Therefore, they are decimated later in the encoding process, and sent earlier during the transmission process. The user can freely select the weights, α, depending on the user's preference on the relative importance of geometry versus texture information. In case of 3D models without texture, only $v(i)$ is considered.

The process of vertex decimation is applied, vertex-by-vertex, from vertices of low $m(i)$ to vertices of high $m(i)$, until only irremovable vertices remain, which form the base mesh. Vertices that satisfy either one of the following two conditions are considered irremovable, so as to retain the topology and the appearance of the simplified model.

1. The triangles surrounding the vertex do not form a closed ring. Vertices on the boundaries satisfy these conditions, so this condition prevents us from decimating vertices on the boundaries.

2. The vertex has extraneous triangles connected to it. If the model has partially junctions of triangles, the vertices connected to them are not decimated.

2.2. Re-Triangulation and Texture Re-Mapping



C : closest vertex

Figure 2: Re-triangulation and texture re-mapping

Once a vertex is decimated, re-triangulation and texture re-mapping are applied to fill up the hole caused by the decimation. As we mentioned above, 3D meshes may have two types of attribute data, vertex attributes and corner attributes [12]. The corner-attributed textures are distributed at random in the texture map. The vertex attributed texture map typically looks like regular images. Thus, we need to consider both cases of corresponding the 3D geometry with the texture map. In case of the vertex attributed texture coordinates, once a vertex is decimated, the textures belonging to the triangles originally connected to the vertex can be re-mapped directly to the base left by the decimation. Since the texture coordinates of the remaining vertices will remain the same, any re-triangulation scheme produces similar results so no texture re-mapping is needed. However in case of the corner attributed texture coordinates, the re-triangulation and the re-mapping significantly affect the decimated model. We now discuss the method we use to correspond each triangle to the texture map. First, among all vertices connected to the vertex that is considered, we find the vertex that has the closest distance from the removed vertex. Then, the removed vertex is mapped to the closest vertex, which results in a triangle fan as in Figure 2. The textures owned by the triangles that are retained are then re-mapped to the new triangles. For example, in Figure 2, the decimated vertex is moved to the vertex **C**. The original textures of triangles 1, 4, 5 and 6 are hence mapped to the new four triangles.

2.3. Compression algorithm

The encoder, following the vertex decimation algorithm aforementioned, replaces the original mesh with a base mesh and a sequence of vertices with associated attribute data. Each vertex is encoded as a seven-tuple: the index of the closest vertex on the edge, the indices of the two vertices on which we start the fan triangulation, the number of triangles to traverse, the x, y, z coordinates of the decimated vertex, and two texture coordinates, s and t, for each vertex in the two triangles removed by the decimation. For 3D models without texture, texture coordinates are not necessary. Vertices in the base mesh are numbered sequentially, and each new vertex is assigned the next available index.

We encode the vertex indices using arithmetic coding. To complete the compression, we encode the vertex coordinates and texture coordinates. Since 3D objects are often well modeled as piecewise smooth regions, the vertex and texture coordinates are highly correlated. These coordinates are predictable by using the neighboring vertices. The vertex coordinates x, y, z are predicted by a linear combination of the neighboring vertices. Similarly, for the models with the vertex attributed texture coordinates, they are predicted by a linear combination of the neighboring texture coordinates.

In case of corner attributed texture coordinates, the texture coordinate of the first vertex is transmitted without any prediction and then the coordinate of the second vertex is predicted by the first coordinate. The coordinate of the third vertex is predicted by the average of the first two coordinates. The residues are quantized by a prescribed step size and then arithmetic coded.

2.4. Coding of Other Attribute Data

The algorithm mentioned above can be easily extended to models with other attribute data such as colors and normals. We incorporate the color and normal differences to the metric (1) in a similar way. We assign the colors and the normals by the same means of the texture re-mapping in Section 2.2. As in case of the texture, the indices and the actual data such as colors and normal vectors are transmitted separately. Once a vertex is decimated, the indices of the data assigned to the vertex are transmitted without any prediction or entropy coding. The encoder codes only the colors and the normal vectors used by the current meshes with prediction and entropy coding.

3. PROGRESSIVE TEXTURE CODING

As is described above, there are two types of texture correspondence to be considered, correspondence for each vertex and for each corner. In the former case, one texture image is mapped to a whole model. Since this type of texture is often as smooth as typical 2D images, we have adopted the wavelet coding algorithm in [13] to encode the texture, resulting in SNR progressive bitstreams. In the latter case, the texture map contains many triangles of various sizes. For this type of texture, we encode each triangle independently and send it to the decoder. The coding method we employ is as follows. First consider a rectangle circumscribing the triangle. We choose a rectangle with each dimension being a multiple of 8. We fill up the pixels outsize the triangle with the pixels at boundary of the triangle by vertical padding followed by horizontal padding, similar to what is used in MPEG-4. Then, we compress the rectangle using discrete cosine transform (DCT), scalar quantization, zigzag scan and Huffman coding, similar to the JPEG algorithm. In our framework, only the textures required to render a current level of the model are transmitted. Hence, progressive texture coding is achieved.

4. EXPERIMENTAL RESULTS

4.1. Compression Results

We tested several 3D models in VRML format downloaded from public web sites. Table 1 shows that comparison between our algorithm and resolution progressive mode of the MPEG-4, which has 10 LOD. Our 3D representation has continuously progress resolution, i.e., the number of LOD corresponds to the number of decimated vertices plus one. In our algorithm, the base mesh of each tested model is compressed by the non-progressive codec of MPEG-4 3D coding algorithm [6].

Quantization of 10-bit resolution is applied to all vertex coordinates. From the results in Table 1, we can see that not only is our representation more "granular," our compression efficiency is also better than the MPEG-4.

4.2. Progressive Streaming

We have also implemented a viewer to progressively display the 3D models coded by our algorithm. Once sufficient bits are downloaded from the server to display more detail, the new model is updated on the display. While the file is being downloaded, the user can change the viewpoint to examine the model. If the user is not interested in the model, the downloading can be stopped at any time. Figure 3 shows what a model looks like as it is being loaded through a 28.8K modem dial-up connection. This model has corner attributed textures. It can be seen that even a low level version of the model gives the user a very good idea of what the complete model would look like. Since we use the texture difference as the measure of importance for vertices, color patterns, such as the edge between white and black feathers in Figure 3(a), are nicely rendered during the whole process.

5. CONCLUSION

In this paper we demonstrated joint geometry/texture progressive coding of 3D models. We have created tools that code 3D files into progressive bitstreams, and a browser that allows the user to download and view these files progressively. Our viewer has been implemented and fully tested. It is available for download at http://amp.ece.cmu.edu/

ACKNOWLEDGEMENTS

This work is partially supported by Japan Society of the Promotion of Science (JSPS).

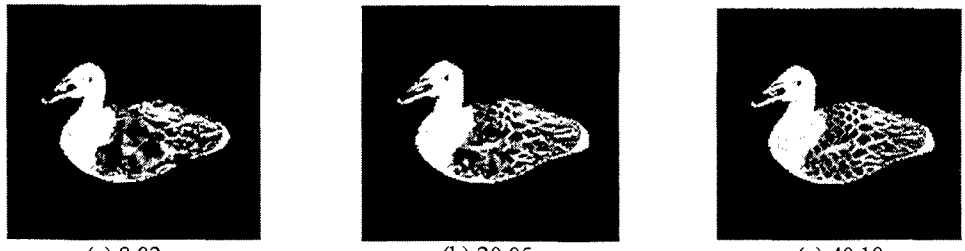
REFERENCES

[1] H. Hoppe, "Progressive Meshes," *Proceedings SIGGRAPH* 96, pp. 99-108. ACM SIGGRAPH, 1996.
[2] M. Garland and P. S. Heckbert, "Surface Simplification using Quadratic Error Metrics," *Proceedings SIGGRAPH 97*, pp. 209-216. ACM SIGGRAPH, 1997.
[3] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes," *Proceedings SIGGRAPH 95*, pp. 173-182, ACM SIGGRAPH 1995.
[4] Web3D Consortium Working Groups, http://www.web3d.org/fs_workinggroups.htm
[5] MPEG-4 SNHC web page, http://www.es.com/mpeg4-snhc
[6] MPEG-4 SNHC, Gabriel Taubin, editor, "SNHC Verification Model 9.0 [3D Mesh Encoding]," W2301, July 1998.
[7] R. Pajarola, and J. Rossignac, "Compressed Progressive Meshes," Tech. Report GIT-GUV-99-05, Georgia Institute of Technology, 1999.
[8] B. Koh and T. Chen, "Progressive VRML Browser," IEEE Intl. Workshop on Multimedia Signal Processing, Sep 1999.
[9] A. Khodakovsky, P. Schroder and W. Sweldens, "Progressive Geometry Compression," preprint, California Institute of Technology, 2000.
[10] J. Li and J. Kuo, "Progressive Coding of 3-D Graphics Models," *Proceedings of the IEEE*, vol. 86, no. 6, June 1998
[11] G. Taubin, J. Rossignac, "3D Geometry Compression," no.21 in Course Notes. ACM SIGGRAPH 1999.
[12] J.D. Foley, *Computer Graphics: Principles and Practice*, Addison-Wesley Pub Co.
[13] A. Said and W. Pearlman, "A New, Fast, and efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243-250. 1996.

Model	#v	#t	#dv	Attributes	Proposed method (bytes)	MPEG-4 hierarchical mode (bytes)	Original VRML (KB) (*1)
Beethoven	2655	5030	2092	none	10080	13951	50
Femur	3897	7798	3824	none	13896	18832	88
Skull	10952	22104	10596	none	45076	59296	257
Triceratops	2832	5660	2762	none	10524	13831	63
Horse	11135	22258	11060	none	40299	48403	266
Duck	5013	10009	4874	texture	147811	- (*2)	605
Vase 1	5153	10044	4685	texture	126082	- (*2)	651
Vase 2	5614	10133	3338	texture	154757	- (*2)	651
Totem pole	5184	10044	3968	texture	160094	- (*2)	683
Totem pole	5184	10044	3969	color	94809	- (*2)	242
Human Face	1221	2374	1084	color	7340	- (*2)	25
Duck	5013	10009	4817	normal	80831	- (*2)	235
Vase	5614	10133	3350	normal	106222	- (*2)	258
Totem pole	5184	10044	3983	normal	92884	- (*2)	242

#v: number of vertices, #t: number of triangles, #dv: number of decimated vertices
*1: Original VRML files are gziped and the texture models include texture files compressed by JPEG.
*2: MPEG-4 Software does not work for these models.

Table 1: Comparison of total number of bytes between the proposed algorithm and MPEG-4



(a) 8.02 sec (b) 20.05 sec (c) 40.10 sec

Figure 3: Simulation of the progressive 3D models under 28.8 kbps environment